Ruby Grant 2018 Mentor report
“Enhancing Ruby’s concurrency tooling”

Developer: Petr Chalupa
Mentor: Koichi Sasada (Cookpad Inc.)

1. Project Evaluation

Concurrency is one of the most important topic Ruby should support more. Petr is a lead maintainer of concurrent ruby maintainer and his contribution was huge for Ruby’s concurrent support, especially on the parallel threading Ruby interpreters such as JRuby and so on.
He achieved the following development topics.

Throttle

A utility to manage concurrency level of tasks. The following example is quoted from the document.

Limiting concurrency level of a concurrently executed block to two

```
max_two = Concurrent::Throttle.new 2
# => #<Concurrent::Throttle:0x000002 capacity available 2 of 2>

# used to track concurrency level
concurrency_level = Concurrent::AtomicFixnum.new
# => #<Concurrent::AtomicFixnum:0x000003 value:0>
job = -> do
  # increase the current level at the beginning of the throttled block
  concurrency_level.increment
  # work, takes some time
  do_stuff
  # read the current concurrency level
  current_concurrency_level = concurrency_level.value
  # decrement the concurrency level back at the end of the block
  concurrency_level.decrement
  # return the observed concurrency level
  current_concurrency_level
end

# create 10 threads running concurrently the jobs
Array.new(10) do
  Thread.new do
```

```
    max_two.acquire(&job)
  end
# wait for all the threads to finish and read the observed
# concurrency level in each of them
end.map(&:value)                        # => [2, 2, 1, 1, 1, 2, 2, 2, 2, 1]
```

Cancellation

A tool to provide cooperative cancellation between concurrent entities. Ruby has several features to interrupt other threads, however they are not safe (or difficult to use safely). “Cancelleation” provide cooperative way to introduce cancellation points. The following example is quoted from the document.

Create cancellation and then run work in a background thread until it is cancelled.

```
cancellation, origin = Concurrent::Cancellation.new
# => #<Concurrent::Cancellation:0x000002 pending>
# - origin is used for cancelling, resolve it to cancel
# - cancellation is passed down to tasks for cooperative cancellation
async_task = Concurrent::Promises.future(cancellation) do |cancellation|
  # Do work repeatedly until it is cancelled
  do_stuff until cancellation.canceled?
  :stopped_gracefully
end
# => #<Concurrent::Promises::Future:0x000003 pending>

sleep 0.01                               # => 0
# Wait a bit then stop the thread by resolving the origin of the
cancellation
origin.resolve
# => #<Concurrent::Promises::ResolvableEvent:0x000004 resolved>
async_task.value!                        # => :stopped_gracefully
```

Channel

FIFO communication channel like Go-language’s channels. The followings are quoted from the documents.

```
# Let's start by creating a channel with a capacity of 2 messages.

ch = Concurrent::Promises::Channel.new 2
# => #<Concurrent::Promises::Channel:0x000002 capacity taken 0 of 2>
# We push 3 messages, then it can be observed that the last
# thread pushing is sleeping since the channel is full.

threads = Array.new(3) { |i| Thread.new { ch.push message: i } }
sleep 0.01 # let the threads run
threads
# => [#<Thread:0x000003@channel.in.md:14 dead>,
#     #<Thread:0x000004@channel.in.md:14 dead>,
#     #<Thread:0x000005@channel.in.md:14 sleep_forever>]
```

Erlang Actors

Actor implementation based on Erlang semantics. The following feature is same as Erlang Actors (quoted from his final report).

- exit behavior (called termination in Ruby to avoid collision with Kernel#exit),
- ability to link and monitor actors,
- ability to have much more actors then threads,
- ordering guarantees between messages and signals,
- message receiving features.

The following example is quoted from documents.

```
# The simplest example is to use the actor as an asynchronous execution.
# Although, Promises.future { 1 + 1 } is better suited for that purpose.

actor = Concurrent::ErlangActor.spawn(type: :on_thread, name: 'addition') { 1
+ 1 }
# => #<Concurrent::ErlangActor::Pid:0x000002 addition terminated normally
with 2>
actor.terminated.value!                  # => 2

# Let's send some messages and maintain some internal state which
# is what actors are good for.

actor = Concurrent::ErlangActor.spawn(type: :on_thread, name: 'sum') do
  sum = 0 # internal state
  # receive and sum the messages until the actor gets :done
  while true
    message = receive
    break if message == :done
    # if the message is asked and not only told,
    # reply with the current sum (has no effect if actor was not asked)
    reply sum += message
  end
  # The final value of the actor
  sum
end
# => #<Concurrent::ErlangActor::Pid:0x000003 sum running>
```

Integration

Not only introduce new concurrent utilities, but he also introduce integration mechanism between them. The following example is quoted from the document and it uses Throttle, ErlangActor and Prommises.

```
throttle = Concurrent::Throttle.new 10
1000.times do
  Thread.new do
    actor = Concurrent::ErlangActor.spawn_actor type: :on_pool,
                                                executor: throttle.on(:io) do
      receive(keep: true) { |m| reply m }
    end
    actor.ask :ping
    Concurrent::Promises.future_on(throttle.on(:fast)) { 1 + 1 }.then(&:succ)
  end
end
```

Documents

He also wrote careful documents which help user to introduce concurrent-ruby to their projects.

Overall evaluation

As described above, he achieved great development of the software and wrote excellent document. I believe this grant project has great success.

His intermediate report and final report is here:
https://github.com/ruby-concurrency/concurrent-ruby/blob/ruby-association/docs-source/ruby-association-intermediate-report.md
https://github.com/ruby-concurrency/concurrent-ruby/blob/ruby-association/docs-source/ruby-association-final-report.md

2. Mentor's position

I reviewed his report and discuss some topics. Also we discussed Guild, another concurrent entity at RubyKaigi 2019.

3. Future prospects

In his proposal, there are other concurrent abstraction and it is not started yet. I hope concurrent-ruby will be extended with these ideas and become good de-fact standard library for Ruby’s threading programs.